Chapter 3: Strings, Vectors and Arrays

3.1 Namespace `using` Declaration

```
#include <iostream>

// using declarations for names from the standard library
using std::cin;
using std::cout;
using std::endl;
int main()
{
   cout << "Enter two numbers:" << endl;

   int v1, v2;
   cin >> v1 >> v2;

   cout << "The sum of " << v1
        << " and " << v2
        << " is " << v1 + v2 << endl;

   return 0;
}
```

Or using entire `std` namespace

```
#include <iostream>

// using declarations for the entire standard library
using namespace std;

int main()
{
   cout << "Enter two numbers:" << endl;

   int v1, v2;
   cin >> v1 >> v2;

   cout << "The sum of " << v1
        << " and " << v2
        << " is " << v1 + v2 << endl;

   return 0;
}
```

(see ppt)

3.2 Library `string` Type

`string` I/O

StringIO.cpp

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
   string s;
   cin >> s;
   cout << s << endl;
   return 0;
}
```

Q: what are the outputs if we enter `Hello World!` from inputs?

A:

Now another example:

StringGetLine.cpp

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
   string line;
   getline(cin, line);
   cout << line << endl;
   return 0;
}
```

Q: what are the outputs if we enter `Hello World!` from inputs?

A:

Quick Checks

StringIOEx1.cpp

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
   string s;
   while (cin >> s)
      cout << s << endl;
   return 0;
```

```
}
```

StringIOEx2.cpp

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
    string s;
    while (getline(cin, s))
        cout << s << endl;
    return 0;
}
```

Q1: What does the above code fragment intend to achieve?

A1:

Q2: How to stop the program?

A2:

Table 3.2 `string` Operation

`os << s`	Writes `s` onto output stream `os`. Returns `os`.
`is >> s`	Reads whitespace-separated string from `is` into `s`. Returns `is`.
`getline(is, s)`	Reads a line of input from `is` into `s`. Returns `is`.
`s.empty()`	Returns `true` if `s` is empty; otherwise returns `false`.
`s.size()`	Returns the number of characters in `s`.
`s[n]`	Returns a reference to the `char` at position `n` in `s`; positions start at 0.
`s1 + s2`	Returns a `string` that is the concatenation of `s1` and `s2`.
`s1 = s2`	Replaces characters in `s1` with a copy of `s2`.
`s1 == s2`	The `strings` `s1` and `s2` are equal if they contain the same characters.
`s1 != s2`	Equality is case-sensitive.
`<, <=, >, >=`	Comparisons are case-sensitive and use dictionary ordering.

`string` `size` Operation:

It might be logical to expect that `s.size()` returns an `int` or an `unsigned`. Instead,

`s.size()` returns a `string::size_type` value. The reason is that the string class—and most other library types—defines several companion types. These companion types make it possible to use the library types in a machine independent manner. The type `size_type` is one of these companion types.

To use the `size_type` defined by `string`, we use the scope operator to say that the name `size_type` is defined in the string class. Although we don't know the precise type of `string::size_type`, **we do know that it is an** `unsigned` **type big enough to hold the size of any string**.

It can be tedious to type `string::size_type`. Under the new standard, we can ask the compiler to provide the appropriate type by using `auto`

```
string s = "I am a C++ string";
string::size_type len1 = s.size(); //C++98
auto len2 = s.size(); // len has type string::size_type, C++11
```

Exercise 3.1 In-class Coding Exercise

Ex31.cpp

Write a program to read strings from the standard input, add their size and merge what is read into one large string. Use ! to terminate the input. Print the input strings, their size, merged string and its size. A sample run looks like:

```
Enter a few strings and terminate with !: I love C++ programming !
I size is: 1
love size is: 4
C++ size is: 3
programming size is: 11
The merged string is: IloveC++programming and its size is: 19
```

Answer:

(see ppt)

3.2.3 Dealing with Characters in a `string`

Table 3.3 `cctype` Functions

`isalnum(c)`	`true` if `c` is a letter or a digit.
`isalpha(c)`	`true` if `c` is a letter.
`iscntrl(c)`	`true` if `c` is a control character.
`isdigit(c)`	`true` if `c` is a digit.
`isgraph(c)`	`true` if `c` is not a space but is printable.
`islower(c)`	`true` if `c` is a lowercase letter.
`isprint(c)`	`true` if `c` is a printable character (i.e., a space or a character that has a visible representation).
`ispunct(c)`	`true` if `c` is a punctuation character (i.e., a character that is not a control character, a digit, a letter, or a printable whitespace).
`isspace(c)`	`true` if `c` is whitespace (i.e., a space, tab, vertical tab, return, newline, or formfeed).
`isupper(c)`	`true` if `c` is an uppercase letter.
`isxdigit(c)`	`true` if `c` is a hexadecimal digit.
`tolower(c)`	If `c` is an uppercase letter, returns its lowercase equivalent; otherwise returns `c` unchanged.
`toupper(c)`	If `c` is a lowercase letter, returns its uppercase equivalent; otherwise returns `c` unchanged.

C++ 11 If we want to do something to every character in a string, by far the best approach is to use a statement introduced by the new standard: the **range for** statement. This statement iterates through the elements in a given sequence and performs some operation on each value in that sequence. The syntactic form is

```
for (declaration : expression)
    statement
```

(see ppt)

Exercise 3.2 In-class Coding Exercise

Ex32.cpp

Write a program to read a line of strings from the standard input and count the number of punctuations in the line. A sample run looks like:

```
Enter a line of strings: Hello, my name is Doris!!!
The number of punctuations is: 4
```

Answer:

(see ppt)

3.3.1 Defining and Initializing `vectors`

Table 3.4 The ways to Initialize a `vector`

`vector<T> v1`	`vector` that holds objects of type `T`. Default initialization; `v1` is empty.
`vector<T> v2(v1)`	`v2` has a copy of each element in `v1`.
`vector<T> v2 = v1`	Equivalent to `v2(v1)`, `v2` is a copy of the elements in `v1`.
`vector<T> v3(n, val)`	`v3` has `n` elements with value `val`.
`vector<T> v4(n)`	`v4` has `n` copies of a value-initialized object.
`vector<T> v5{a,b,c...}`	`v5` has as many elements as there are initializers; elements are initialized by corresponding initializers.
`vector<T> v5 = {a,b,c...}`	Equivalent to `v5{a,b,c...}`.

Table 3.5 `vector` Operation

`v.empty()`	Returns `true` if `v` is empty; otherwise returns `false`.
`v.size()`	Returns the number of elements in `v`.
`v.push_back(t)`	Adds an element with value `t` to end of `v`.
`v[n]`	Returns a reference to the element at position `n` in `v`.
`v1 = v2`	Replaces the elements in `v1` with a copy of the elements in `v2`.
`v1 = {a,b,c...}`	Replaces the elements in `v1` with a copy of the elements in the comma-separated list.
`v1 == v2` `v1 != v2`	`v1` and `v2` are equal if they have the same number of elements and each element in `v1` is equal to the corresponding element in `v2`.
`<, <=, >, >=`	Have their normal meanings using dictionary ordering.

Using `push_back` member function

To store a value in a `vector` that does not have a starting size or that is already full, you should use the `push_back` member function. This function accepts a value as an argument and store it in a new element placed at the end of vector. It "pushes" the value at the "back" of the `vector`. For example,

```
vector<int> x;
x.push_back(12);
```

Q: what happens?

A:

With introduction of `string` and `vector`, we can easily store word from standard input into the `vector` container and process these words upon request. For example, we can ask users to input a few words, store them in a `vector` and parse and print those words that are longer than 4 characters.

```
I think this is my mouse.
^Z
The words longer than 4 characters are: think mouse.
請按任意鍵繼續 . . .
```

VectorStringEx.cpp

```
#include <iostream>
#include <string>
#include <vector>
using namespace std;

int main()
{
    string word;
    vector<string> text;
    while (cin >> word)
        text.push_back(word);
    for (auto s : text)
        if (s.size() > 4) cout << s << endl;
    return 0;
}
```

Exercise 3.3 In-class Coding Exercise

Ex33.cpp

Read a sequence of words from `cin` and store the values a `vector`. After you've read all the words, process the vector and change each word to uppercase. Finally, print the transformed elements, eight words to a line. A sample run looks like:

```
Enter a few words and use ! to terminate your input: You learn C++ because you w
ant to become a better programmer !
YOU LEARN C++ BECAUSE YOU WANT TO BECOME
A BETTER PROGRAMMER
Process returned 0 (0x0)   execution time : 59.639 s
Press any key to continue.
```

Answer: